METHOD AND SYSTEM FOR STARTING OPERATING SYSTEM ENVIRONMENT

Patent number: JP6324849

Publication date: 1994-11-25

Inventor: RAWSON III FREEMAN L; SOTOMAYOR GUY G JR

Applicant: INTERNATL BUSINESS MACH CORP <IBM>

Classification:

- international: G06F9/06; G06F9/445

- european:

Application number: JP19940107472 19940425

Priority number(s):

Also published as:

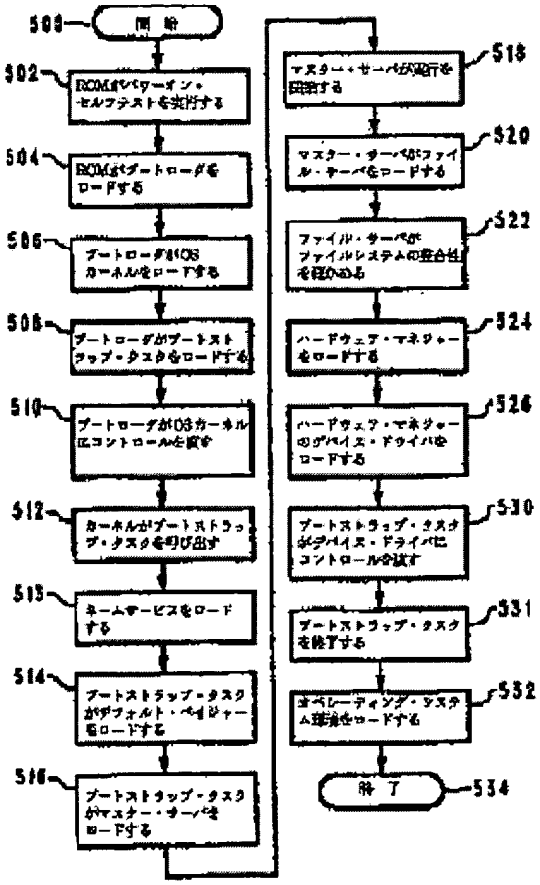
EP0622731 (A2)

EP0622731 (A3)

Abstract of **JP6324849**

PURPOSE: To provide an operating system environment.

CONSTITUTION: In a data processing system having an OS kernel and plural device drivers separated from the OS kernel, steps for (1) preparing a boot volume having the plural files of data structure constituted from plural files on a storage device and parameter for identifying the positions in a memory and (2) for loading plural programs related to the initialization of the operating system environment by using the boot volume and initializing the operating system environment are provided.



Reference 3

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平6－324849

(43)公開日　平成６年(1994)11月25日

(51)Int.Cl.[5]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	9/06	４１０　Ｄ	9367－5B			
	9/445					
			9367－5B	Ｇ０６Ｆ　9/06	４２０　Ｇ	

審査請求　有　請求項の数12　ＦＤ　(全 15 頁)

(21)出願番号　特願平6－107472

(22)出願日　平成６年(1994)４月25日

(31)優先権主張番号　０８／０５４，１１７

(32)優先日　1993年４月26日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州　アーモンク　(番地なし)

(72)発明者　フリーマン　エル.　ローソン、サード
アメリカ合衆国　33487－2242　フロリダ州　ボカラトン　ウッドビューテラス　17762

(74)代理人　弁理士　合田　潔　(外３名)

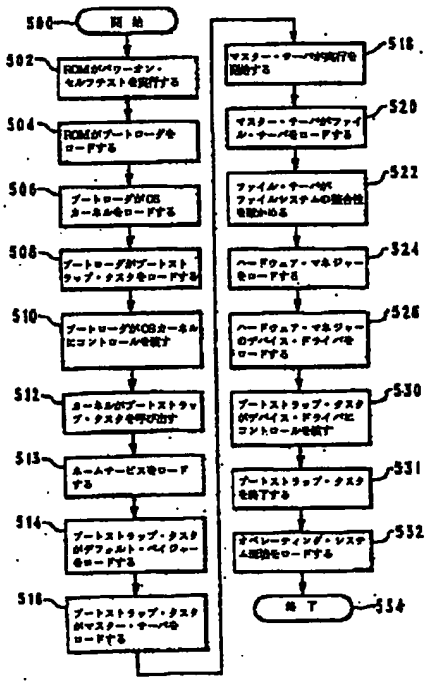
最終頁に続く

(54)【発明の名称】　オペレーティング・システム環境の起動方法およびシステム

(57)【要約】

【目的】　オペレーティング・システム環境を起動する方法を提供する。

【構成】　ＯＳカーネルと、ＯＳカーネルから分離されている複数のデバイス・ドライバを持つデータ処理システム上で、(1)記憶装置上の複数のファイルから構成されるデータ構造の複数のファイルおよびその位置を識別するパラメータを有するブート・ボリュームをメモリに用意し、(2)上記ブート・ボリュームを使用して、オペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードして、オペレーティング・システム環境の初期化を行う、ステップを有するオペレーティング・システム環境起動方法。



1

【特許請求の範囲】

【請求項1】　カーネルと上記カーネルから分離されている複数のデバイス・ドライバ・プログラムを備えたデータ処理システムでオペレーティング・システム環境を起動させる方法であって、
記憶装置上の複数のファイルから構成されるデータ構造の複数のファイルおよびその位置を識別するパラメータを有するブート・ボリュームをメモリに用意し、
上記ブート・ボリュームを使用して、オペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードして、オペレーティング・システム環境の初期化を行う、
ステップを有する方法。

【請求項2】　上記パラメータが、
上記ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、
上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を有し、ファイルを構成するデータ構造が少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされる請求項1に記載の方法。

【請求項3】　カーネルと上記カーネルから分離されている複数のデバイス・ドライバ・プログラムを備えたデータ処理システムでオペレーティング・システム環境を起動させる方法であって、ブート・ボリュームをメモリに用意するステップを有し、上記ブート・ボリュームが
上記ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、
上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を備え、ファイルを構成するデータ構造が少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされ、さらに上記方法が、
上記ブート・ボリュームを使用してオペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードする、
ステップを有し、オペレーティング・システム環境の初期化が行われる方法。

【請求項4】　上記のロードするステップが、デバイス・ドライバをロードすることを含む請求項3に記載の方法。

【請求項5】　マスター・サーバをロードすることをさらに含む請求項4に記載の方法。

【請求項6】　ファイル・サーバおよびファイル・システムをロードすることをさらに含む請求項5に記載の方法。

2

【請求項7】　上記ファイル・システムがログと複数のファイルを有し、上記ファイルを使う前に、上記ファイルに訂正が必要であるか否かを判断するためにログを使用することをさらに含む請求項6に記載の方法。

【請求項8】　デバイス・ドライバをロードした後に、オペレーティング・システム環境をロードすることをさらに含む請求項7に記載の方法。

【請求項9】　オペレーティング・システム環境と、ＯＳカーネルと、複数のデバイス・ドライバを持ち、上記複数のデバイス・ドライバがカーネルと分離されているデータ処理システムであって、
デバイス・ドライバ機能を除くカーネルと、
記憶装置に記憶されているブート・ボリュームと、
ブートストラップ・タスクと、
を有し、上記ブート・ボリュームが、
ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、
上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を有し、ファイルを構成するデータ構造が、少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされ、さらに上記ブートストラップ・タスクが上記ブート・ボリュームを使用してデータ記憶装置からファイルをメモリにロードすることによりオペレーティング・システム環境が初期化されるシステム。

【請求項10】ブート・ボリュームにより保証されて、データ処理システムが正しくない可能性のあるファイルでなく、正確なファイルを使用することができる請求項9に記載のシステム。

【請求項11】オペレーティング・システム環境と、ＯＳカーネルと、複数のデバイス・ドライバを持ち、上記複数のデバイス・ドライバが上記カーネルと分離されているデータ処理システムであって、
記憶装置上の複数のファイルから構成されるデータ構造の複数のファイルおよびその位置を識別するパラメータを有するブート・ボリュームをメモリに用意する手段と、
上記ブート・ボリュームを使用して、オペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードする手段と、
を有することによりオペレーティング・システム環境の初期化を行うシステム。

【請求項12】上記パラメータが、
ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、

上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を有し、ファイルを構成するデータ構造が、少なくとも１つの上記ブロック・リスト・エレメントを使用して識別されアクセスされる、請求項１１に記載のシステム。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、マイクロ・カーネルを使用してデータ処理システムを初期化する方法とシステムに関する。

【０００２】

【従来の技術】オペレーティング・システムは、データ処理システムのオープンシステム標準において、優劣を競う分野になっている。国内・海外を問わず業界におけるこの分野の競争の結果、１つの統合的なオペレーティング・システムが複数のオペレーティング・システムをサポートすることが望ましいとされるようになった。

【０００３】オペレーティング・システムを作成する１つの手法に、基本的なハードウェア資源を制御するオペレーティング・システムの部分と、オペレーティング・システム環境のユニークな特徴を定義する部分とを分離して、「純粋なカーネル」を作る方法がある。ハードウェア資源に関連する部分はオペレーティング・システム・カーネル（以下ＯＳカーネルまたはカーネルと略す）と呼ばれることが多い。このような設計を採用すると、１つのデータ処理システムの上で複数のオペレーティング・システムを動かすことが可能になる。また、同一のデータ処理システム上で、複数のオペレーティング・システムをネイティブモードで同時に動かすことが可能になる。

【０００４】しかし、純粋なカーネルを作成するのに複雑な問題は、既にあるオペレーティング・システムに依存せず、かつ、純粋なカーネルが異種のオペレーティング・システム環境をサポートするようにすることである。例えば、オペレーティング・システムの純粋カーネルは、いくつかの異種のオペレーティング・システム環境に対して中立であり、そのカーネル（即ちマイクロ・カーネル）をブートストラップするためにユーザ・レベルでデバイス・ドライバを使用でき、かつ、その上に少なくとも１つのオペレーティング・システムを作動させなければならない。

【０００５】現在、この問題の解決は、ＯＳカーネルとそのオペレーティング・システムで実行されるプログラムとを結びつけるか、あるいは、システムが、ブートのすぐ後でロードされるオペレーティング・システム環境を持ち、ブートストラップをしている間に使用できるファイル・システムを持っていると仮定することで対処している。しかし、デバイス・ドライバとブートストラップ中あるいはその後でロードされるタスクとが分離されている環境では、この解決策は動かない。

【０００６】

【発明が解決しようとする課題】したがって、ハードウェアで定義したROMブートの終了から、最初のオペレーティング・システム環境の標準的な初期化の実行までの連続的な手順の全てに対処できるブートストラップのアーキテクチャが求められる。

【０００７】

【課題を解決するための手段】本発明は、マイクロ・カーネルを使用してデータ処理システムを初期化する方法とシステムを提供することを目的とする。

【０００８】本発明は、オペレーティング・システム環境と、ＯＳカーネルと、カーネルから分離されている複数のデバイス・ドライバ・プログラムを持っているデータ処理システムに関するものである。このシステムでは、カーネルはデバイス・ドライバ機能を持たず、デバイス・ドライバはユーザ・レベルのタスク即ちプログラムである。

【０００９】記憶装置にブート・ボリュームが用意され、ブート・ボリュームはファイル・アンカーと、複数のファイルヘッダと、複数のブロック・リスト・エレメントを持っている。ファイル・アンカーはブート・ボリュームに関連するファイルを探すための始まりのポイントを示し、ファイル・ヘッダーはアンカーに関連し、それぞれがブート・ボリュームに関連しているファイルの識別を持ち、ブロック・リスト・エレメントはそれぞれ記憶装置の中の１つの連続したデータのブロックを識別し、ファイルを構成するデータ構造が、少なくとも１つのブロック・リスト・エレメントを使用して識別されアクセスされる。ブートストラップ・タスクにより、データ記憶装置からメモリに、ブート・ボリュームを使用してファイルがロードされ、ファイルがアクセスできるようになり、オペレーティング・システム環境が初期化される。

【００１０】

【実施例】図１に、本発明が実施できるデータ処理システムを示す。コンピュータ５０が、システム・ユニット５２、ディスプレイ装置５４、キーボード５６、およびマウス５８を備えている。コンピュータ５０は、IBM PS/2あるいはIBM RISC SYSTEM/6000等あるいは同等の、本発明の実施に適したシステムである。

【００１１】図２に、本発明が実施できるコンピュータ５０の主要な構成要素を示す。システムユニット５２は各種の構成要素を相互接続するためのシステムバス６０を持つことが望ましい。マイクロプロセサ６２はシステムバス６０（例えばマイクロチャネル・システムバス）に接続され、また、数値計算用のコプロセサ６４を持ってもよい。また、ダイレクト・メモリ・バス（DMA）

コントローラ６６がシステムバス６０に接続され、これにより、大量の入出力データ転送中に、各種のデバイスがマイクロプロセサ６２からのサイクル割り当てが可能になる。

【００１２】ＲＯＭ６８とＲＡＭ７０もシステムバス６０に接続されている。ＲＯＭ６８はパワー・オンの際のセルフテストを含んでいる。ＣＭＯＳ　ＲＡＭ７２がシステムバス６０につながり、システム構成情報を持つことも可能である。

【００１３】また、システムバス６０にはメモリ・コントローラ７４、バス・コントローラ７６、インタラプト・コントローラ７８が接続されていて、各種の周辺装置、アダプタおよびデバイス間のデータの流れをシステムバス６０を介して制御する。システムユニット５２は、さらに各種の入出力コントローラ、即ち、キーボード／マウス・コントローラ８０、ビデオ・コントローラ８２、並列コントローラ８４、直列コントローラ８６、ディスケット・コントローラ８８を持っている。キーボード／マウス・コントローラ８０は、キーボード９０とマウス９２のためのハードウェア・インターフェースである。ビデオ・コントローラ８２は、ディスプレイ装置９４のハードウェア・インターフェースである。並列コントローラ８４は、印刷装置９６のような装置のハードウェア・インターフェースである。直列コントローラ８６は、モデム９８のような装置のハードウェア・インターフェースである。ディスケット・コントローラ８８は、フロッピーディスク・ユニット１００のハードウェア・インターフェースである。ハードディスク・ユニット１０４のインターフェースを提供するディスク・コントローラ１０２のような拡張カードをシステムバス６０に加えてもよい。空きスロット１０６を使って、システム５２に他の周辺装置やアダプタをつけ加えることもできる。

【００１４】図２に示したハードウェアは、具体的な用途によって変わり得る。例えば、既に述べたハードウェアに加えまたはその代わりに、光ディスク媒体、音声アダプタ、PALやEPRMのようなプログラム用のチップ等の周辺デバイスを用いることもできる。

【００１５】図３に、本発明を用いてデータ処理システムをブートする方法のフローチャートを示す。ブート・ボリューム、ブートストラップ・タスク、ＯＳカーネル、マスター・サーバ等の要素は図４から図１１で詳しく述べる。プロセスはブロック５００で始まりブロック５０２に進む。ブロック５０２はパワーオン・セルフテストを行うROMの機能である。ブロック５０４はブート・ローダをロードするROMの機能を示す。ブロック５０６はＯＳカーネルをロードするブート・ローダを示す。ブロック５０８はブートストラップ・タスクをロードするブート・ローダを示す。

【００１６】ブロック５１０で、ブート・ローダはＯＳカーネルにコントロールを渡す。ブロック５１２で、ＯＳカーネルがブートストラップ・タスクを呼び出す。ブロック５１３でネーム・サービスがロードされる。ブロック５１４で、ブートストラップ・タスクがデフォルト・ペイジャーをロードする。次のブロック５１６で、ブートストラップ・タスクがマスター・サーバをロードする。プロセスは次に、マスター・サーバが実行を始めるブロック５１８に進む。ブロック５２０で、マスター・サーバがファイル・サーバをロードする。次に、ブロック５２２で、ファイル・サーバがファイルシステムの整合性を確かめる。次にプロセスは、ハードウェア資源マネジャをロードするブロック５２４に進む。さらに、ハードウェア資源マネジャのデバイス・ドライバをロードするブロック５２６に進む。

【００１７】次にプロセスは、ブロック５３０で、ブート用デバイスのコントロールを、ブートストラップ・タスクから、ブロック５２６でロードされたデバイス・ドライバに渡す。ブートストラップ・タスクはブロック５３１で終了する。その後、プロセスは、ＯＳ環境をロードして初期化するブロック５３２に進む。プロセスはこの後ブロック５３４で終了する。

【００１８】ブート・プロセスでは、データ処理システムのオペレーティング・システムを起動するために、いくつかの異なるブート要素が使用される。これらのブート要素には、ブート・ボリューム、ブート・ローダ、ＯＳカーネル、ブートストラップ・タスク、ネーム・サービス、デフォルト・ペイジャー、イニシャル・ページ・スペース、マスター・サーバ、ファイルシステム・サーバ、ハードウェア資源マネジャ、および、ブート・デバイスドライバのＯＳ環境がある。ブート・ボリュームは、ブート・プロセスでどのファイルやプログラムをロードすべきかを示す記述を持っている。ブート・ボリュームはディスク上の一連の逐次的セクターからなっている。本発明では、ブート・ボリュームはロードすべきファイルの記述だけを持ち、ファイルそのものは持たない。ブート・ボリュームにファイルを持たないことにより、ファイル内容を持った場合にくらべて、ブート・ボリュームを小さくできる。さらに、ブート・ボリュームの内容を、標準的なユーティリティー・プログラムをどれか１つのＯＳ環境で実行して容易に変更できる。こうすることにより、ブート・ボリュームの内容を変更し、いまの内容よりも大きなスペースが要る場合に、ブート・ボリュームに十分なスペースを割り当てることができる。

【００１９】今日、多くのファイル・システムはログを持っていて、システム故障の際のデータの損失を防ぐようにしている。故障したファイルシステムのデータはアクセスできないようにし、ユーティリティー・プログラムがログを適用しファイルシステムを初期化し、データ構造を直してからファイルシステムが使えるようにして

いる。ブート・ボリュームは、ファイルシステムと修正ユーティリティー・プログラムを実行するのに十分なもう１組のデータ構造を持ち、これにより、ファイルシステム動作回復を可能にしている。

【００２０】あるシステムでは、ブート・ボリュームが、ブート・ボリュームに記述されている全てのファイルを持っている場合もあろう。どのように具体化するかは、ブートするときにハードウェアがどのように作動するかに依存している。例えば、RS/6000では、ハードウェアの構成が完了するまで入出力動作は行われない。

【００２１】ブート・ボリュームで記述されているファイルをブート・ボリュームが持っていないシステムでは、本発明を使用すれば、通常の動作でこれらのファイルを読むユーティリティー・プログラムやＯＳ環境プログラムが、故障して正しくない可能性のあるファイルでなく正しいファイルを読むことができる。

【００２２】図４に、本発明によるブート・ボリュームの構造のダイアグラムを示す。ブート・ボリューム２９８には通常レコードと呼ばれるデータ構造、即ち、ブート・ボリューム・ファイル・アンカー３００、ファイルヘッダ３０２、および、ブロック・リスト・エレメント３０６がある。実際のファイルはファイルシステムに記憶され、ブート・ボリュームの設計とレイアウトはファイルシステムに依存しない形で作られる。

【００２３】図４に矢印で示しているが、通常ブート・ボリューム内容は全て同時に作成され、個々のデータ構造はリンクされたリストとしてよりも、アレイとして記憶される。ブート・ボリュームにある全てのオフセットは、ブート・ボリュームの始めからの相対的なバイトオフセットである。また、本発明により、全ての複数バイトの値は、データを解釈するシステムのエンディアンそのままのエンディアン（natural endian）として記憶される。

【００２４】図５に、ブート・ボリューム・ファイル・アンカー３００のダイアグラムを示す。ブート・ボリューム・ファイル・アンカー３００は、ブート・ボリューム２９８の内容を総合したアンカーとしてのレコードである。アンカーは、ブート・ボリューム２９８で記述されたファイルを探すためのスタートポイントの役をする。アンカー３００にはレコードのタイプを記述するフィールド３１０があり、フィールド３１０はこのレコードをブート・ボリューム・ファイル・アンカーとして識別する。フィールド３１２は、ブート・ボリューム・ファイル・アンカー３００が占めるバイト数を表す。次のフィールド３１４は、ブート・ボリューム・ファイル・アンカーの現在のバージョン番号を示す。このフィールドが、ブート・ボリューム・ファイル・アンカー３００のフィールドの構造や意味の変更を表す。

【００２５】フィールド３１６は、ブート・ボリュームが占めるバイト数を表す。バイト数はブート・ボリューム・ファイル・アンカーを含む。フィールド３１８はブート・ボリュームの現在のバージョン番号を示し、ブート・ボリュームのフィールドの構造や意味の変更を表す。フィールド３２０は、ブート・ボリューム・ファイル・アンカーに含まれているファイルヘッダの数を表す。フィールド３２２はそれぞれブート・ボリュームが参照している個々のファイルを記述している。フィールド３２２については後で詳しく述べる。

【００２６】図６に、ファイルヘッダ３２２のダイアグラムを示す。ファイルヘッダは、ブート・ボリュームで使われるデータ構造で、データ処理システムに実際のファイルシステムがロードされる前に、ブート・プロセス中に参照されるそれぞれのファイルを記述している。ファイルヘッダ３２２にフィールド３３０があり、このレコードがファイルヘッダ３２２であると識別するフォーマットを示す。フィールド３３２は、ファイルヘッダが占めるバイト数を示すフィールドである。フィールド３３２の数は、最初のフィールドからブロック・リスト・エレメントの最後までを含む。

【００２７】フィールド３３４は、ファイルヘッダのバージョン、即ち、ファイルヘッダ構造の現在のバージョン番号を示し、ファイルヘッダ３２２のフィールドの構造や意味の変更を表す。フィールド３３６は、ファイルが占める全バイト数、即ちファイルの大きさを示す。フィールド３３８はファイルの名前を示し、ナル（null）で終わるストリングである。典型的には、フィールド３３８には、実際のユーザのファイルシステムの中のファイルの名前が入る。フィールド３３８の大きさは通常256キャラクタである。フィールド３４０は、ファイルヘッダ３２２で記述されたファイルのタイプを表す。以下のファイルタイプがフィールド３４０に記述される。

【００２８】(1)ＯＳカーネル。ファイルがＯＳカーネルであることを示すフラッグ。これにより、ブート・ローダ・プログラムはこの属性を持ったファイルを探し、ＯＳカーネルの名前に永久に符号化しないようにする。

【００２９】(2)ブートストラップ・タスク。ファイルがブートストラップ・タスクであることを示すフラッグ。これにより、ブート・ローダ・プログラムはこの属性を持ったファイルを探し、ブートストラップ・タスクの名前に永久に符号化しないようにする。

【００３０】(3)ネームサービス。ファイルがネームサーバであることを示すフラッグ。これにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、ネームサービス・プログラムの名前に永久に符号化しないようにする。

【００３１】(4)ペイジャー。ファイルがデフォルト・ペイジャー（即ち、まだ誰のものでもないメモリへのペイジャー）であることを示すフラッグ。これにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、ペイジャー・プログラムの名前に永久に符号化し

ないようにする。

【００３２】(5)ペイジング・ファイル。ファイルがペイジング用のデフォルト・ペイジャーによって使われることを示すフラッグ。これにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、ペイジング・ファイルの名前に永久に符号化しないようにする。

【００３３】(6)マスター・サーバ。ファイルがマスター・サーバであることを示すフラッグ。マスター・サーバは他のサーバをロードするプログラムである。このフラッグにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、マスター・サーバの名前に永久に符号化しないようにする。

【００３４】(7)マスター・サーバ構成ファイル。このファイルが、ブート・プロセスの１部として立ち上げるために、マスター・サーバが、システムと他のサーバを構成するのに必要な構成情報を持っていることを示すフラッグ。これにより、ブートストラップ・タスクはファイルを探し、そのファイルを、マスター・サーバが実行を始めるときにマスター・サーバに渡す。

【００３５】(8)実行可能ファイル。ファイルが一般的な実行可能ファイルであることを示すフラッグ。マスター・サーバはこのフラッグを使って、ファイルがロードされるべきサーバかどうかを判断する。

【００３６】(9)データファイル。ファイルが実行可能でないデータを持っていることを示すフラッグ。

【００３７】フィールド３４２は次のファイルヘッダへのオフセットであり、ブート・ボリュームの中にある次のファイルヘッダのブート・ボリュームの始めからのオフセットを示す。このファイルヘッダがブート・ボリュームの中で最後のものである場合は、このフィールドは、本発明の具体化では、ゼロを持つ。フィールド３４４は、ファイルを表すのに必要なブロック・リスト・エレメントの数を示す。フィールド３４６は、ブロック・リスト・エレメントを示し、ディスク上で物理的に連続しているブロックの数を記述する。これについては、後で詳しく述べる。

【００３８】図７に、ブロック・リスト・エレメント３４６を示す。ブロック・リスト・エレメント３４６は、ハード・ディスク・ドライブの上の、物理的に連続したブロックの一群を記述するデータ構造である。ブロック・リスト・エレメント３４６にはフィールド３４８があり、レコードのタイプ即ちレコードのフォーマットを示す。フィールド３４８は、このレコードがブロック・リスト・エレメント３４６の中の１つのブロック・リスト・エレメントであると識別する。フィールド３５０は、ブロック・リスト・エレメント３４６が占めているバイト数を表す。バイト数は最初のフィールドから数えられる。フィールド３５２は、ブロック・リスト・エレメントの最新のバージョン番号を示し、ブロック・リスト・エレメント３４６のフィールドの構造と意味の変更を表す。フィールド３５４は、ディスクドライブ上の１つのブロックが持っているバイト数、即ちブロックの大きさを表す。

【００３９】デバイス識別符号（以下ＩＤ）はフィールド３５６で示される。このフィールドは、デバイスが載っているデバイスを示し、ナルで終わる一連のストリングで表される。フィールド３５８は、ブロック・リスト・エレメント３４６が表すデバイス上での物理的に連続しているブロックの数を示す。フィールド３６０はオフセット・ポインターの大きさを示す。このフィールドは、デバイスフィールドの始めからのオフセットに含まれているビット数を表す。本発明の具体化では、オフセットの記憶に使われるバイト数はいつも４の倍数であるが、オフセットフィールドの全てのビットが意味を持つわけではない。フィールド３６２はデバイスの始めからのオフセットを示す。このフィールドは、このレコードで記述されている最初のブロックのデバイスの始めからのバイトオフセットを持っている。このフィールドの大きさはフィールド３５８で定義される。

【００４０】ブート・ローダは、例えばROMのようなハードウェアあるいはシステムで定めた仕組みによってロードされるプログラムである。ブート・ローダはブート・ボリュームのフォーマットを解釈し、システム・メモリにＯＳカーネルとブートストラップ・タスクをロードする任務を持つ。また、ブート・ローダはＯＳカーネルの実行をスタートさせる。ＯＳカーネルとブートストラップ・タスクをメモリにロードするのに加えて、ブート・ローダは、コントロールをＯＳカーネルに渡す前に、ＯＳカーネルに以下に述べる情報を渡す。

【００４１】図８に、ＯＳカーネルに情報を渡す方法のフローチャートを示す。プロセスはブロック４００で始まり、ブロック４０２に進み、ＯＳカーネルにメモリマップ情報を渡す。メモリマップ情報は、ＯＳカーネルに対して、メモリの物理的なレイアウトを記述するものである。メモリ全部が物理的に連続していないならば、メモリマップは異なるいくつかのメモリ領域で表される。メモリマップは、ＯＳカーネルにとって必要な、領域の属性情報を持つ。

【００４２】プロセスはブロック４０４に進み、メモリの中のブートストラップ・タスクの位置（物理的なアドレス）と大きさがＯＳカーネルに渡される。ＯＳカーネルに渡される情報は、実行が始まるブートストラップ・タスクへのオフセットを含むこともある。さらに、ブートストラップ・タスクがいくつかの領域（即ちテキストとデータ）に分割されている場合には、それぞれの領域を記述する情報もＯＳカーネルの送られる。次にプロセスはブロック４０６に進み、ブート・デバイスのデバイスＩＤがＯＳカーネルに渡される。ブート・デバイスのデバイスＩＤは、ブート・ローダがＯＳカーネルとブートストラップ・タスクをロードしたブート・ボリューム

が入っているデバイスを指す。この情報がＯＳカーネルに提供されるのは、さらにこの情報をブートストラップ・タスクに渡すためである。

【００４３】次に、プロセスはブロック４０８に進み、ＯＳカーネルに、ブート・ボリュームのブート・デバイスのオフセットが渡される。オフセットは、ブート・デバイス上のブート・ボリュームの位置を教えるものである。この情報により、ブート・デバイスの正確なレイアウトがわからなくても、ブート・デバイス上の複数のブート・ボリュームおよびブート・プロセスの他の構成要素が実行できるようになる。この情報も、ＯＳカーネルに渡され、さらにブートストラップ・タスクに渡される。プロセスはブロック４１０で終了する。

【００４４】情報は、公知の方法により、メモリの中のいろいろなプログラムの間で渡される。例えば、パラメータをスタックすること、即ち、メモリの中の情報を指すポインターを、その情報が行くべきプログラムに渡すことによって行える。

【００４５】本発明の具体化により、情報は、ＯＳカーネルに、ニューライン・キャラクター（即ち＼n）で分離された連続したキャラクターのアレイとして提供できる。アレイ全体はナルキャラクターで終わる。アレイのフォーマットは以下のようになる。

【００４６】variable1 = value 1 ＼ n
variable2 = value 2 ＼ n
・・・
variablen = value n ＼ n ＼ 0

変数（variables）の値には以下のタイプがある。

【００４７】

値のタイプ	シンタックス	例
INTTYPE	int	VAR = -1 または VAR = 1
UINTTYPE	uint	VAR = 1
CHARTYPE	char	VAR = c
STRINGTYPE	"..."	VAR = "abcdefg"
MULTIRANGETYPE	int1..int2	VAR = 1..4
	int1,int2	VAR = 1,3
		VAR = 1..4,8..10,12,17

ＯＳカーネルに渡される実際のパラメータを以下に示す。

【００４８】MEMORYMAP = x1..y1, x2..y2, x3..y3

これはシステムのメモリマップを記述している。複数の領域があり得るのは、多くのシステムの場合、メモリがある物理的なアドレスには「空き」があるからである。システムに存在するメモリを十分に記述するのに必要な領域が記述される。値は整数で与えられる。計算の基になる数は十進法である。値を"0x"で始めれば、値はヘクサデシマルとみなされる。

【００４９】BOOTTASK_NUM_REGION = n

これはＯＳカーネルに与えられた領域の数を表しブートストラップ・タスクを記述するものである。値は整数である。

【００５０】BOOTTASK_REGION = x..y, vs, "..."

これはブートストラップ・タスクのアドレス・スペースの個々の領域を表し、値は以下の意味を持つ。

【００５１】x...yはメモリの物理的領域を記述する。値は整数である。計算の基は十進法である。値を"0x"で始めれば、ヘクサデシマルとみなされる。

【００５２】vsは領域のバーチュアル・スタート・アドレスである。ブートストラップ・タスクのアドレス・スペースの中の、領域の始まりを示すアドレスである。値は整数で十進法である。"0x"で始まれば、ヘクサデシマルとみなされる。

【００５３】"..."は領域に適用されるべきプロテクション・フラッグで、一連のストリングで表される。ストリングの内容はフラッグのシンボリック・ネームで、"I"の字で分離されている。シンボリック・ネームは以下のものがある。

VM_PROT_READ	読み取りアクセスができる。
VM_PROT_WRITE	書き込みアクセスができる。
VM_PROT_EXECUTE	実行アクセスができる。

【００５４】BOOTTASK_STATE = n1, n2, ...nm

これはＯＳカーネルがブートストラップ・タスクの実行を始めたときにセットされる実行状態を示す。これによって表されるパラメータは、正しい実行状態を確立するための、使用するプロセサに依存した情報である。

【００５５】KERNEL_PATCH = x..y, value

これにより、ＯＳカーネルの初期化中に、あるカーネルデータを変更することができる。例えば、デバッグのあるオプションを選択してオンにできる。値は以下に説明する。

【００５６】x..yは変更したいカーネルのバーチュアル・アドレスの範囲を表す。範囲はバイトで表すので、範囲としては複数のバイト値で指定しなければならない。範囲が何を表しているかは、値（以下に示す）のタイプによって解釈される。

【００５７】valueはカーネルの複数のアドレスの範囲の中に置かれる値を表す。値は１つの値に限らず、全ての値が同じタイプであればリストにしてよい。

【００５８】BOOT_DEVICE = "..."

これは、ブート・ボリュームが載っているデバイスを表し、デバイスの名前を含んだ一連のストリングである。

【００５９】BOOTVOL_OFFSET = n
これは、ブート・デバイスの始まりからのブート・ボリュームの位置を表す。値は整数で十進法である。"0x"で始まれば、ヘキサデシマルとみなされる。
【００６０】本発明では、ブートストラップ・タスクは以下の３つの主たる機能を持つ。即ち、(1)マスター・サーバも含めて、最初のいくつかのタスクをシステムにロードすること、(2)正式のシステム・レベルの機能がロードされるまで、暫定的なデバイス・ドライバ・セットの役をすること、(3)ブート・ボリュームのファイル・システム・サーバとして機能し、ブート・ボリュームにアクセスする必要のある他のタスクがブート・ボリューム自体の内容を解釈しなくてもすむようにすること。
【００６１】情報はＯＳカーネルからブートストラップ・タスクに渡される。これは２つの方法により行われる。第１の方法は、情報をパラメータ（ブートストラップ・タスクが始まるときにスタックされている）として渡すことである。第２の方法は、情報をＯＳカーネルのブート・パラメータ・リストに載せ、ブートストラップ・タスクが取り出せるようにすることである。
【００６２】図９のフローチャートに、ＯＳカーネルからブートストラップ・タスクに情報を渡す方法を示す。プロセスはブロック４２０で始まり、ブロック４２２に進み、ブート・デバイスのデバイスＩＤがパラメータとしてブートストラップ・タスクに渡される。デバイスＩＤは、ロードされたＯＳカーネルとブートストラップ・タスクの入っているデバイスを識別するものである。
【００６３】プロセスはブロック４２４に進み、ブート・デバイス上のブート・ボリュームがブートストラップ・タスクに渡される。ブート・ボリュームのオフセットは、ブート・デバイス上のブート・ボリュームの位置を示す。これにより、ブートストラップ・タスクはブート・ボリュームのフォーマット情報だけを持てばよく、ブート・デバイスの内容全体の情報を持たなくてすむ。プロセスはブロック４２６で終了する。
【００６４】図１０に示すように、ブートストラップ・タスクは、他に３つのプログラムをロードする。プロセスはブロック４３０で始まり、ブロック４３２に進み、システム・ネーム・サービスがロードされる。ブロック４３４で、デフォルト・ページャーがメモリにロードされる。ブロック４３６でマスター・サーバがメモリにロードされる。プロセスはブロック４３８で終了する。
【００６５】システム・ネーム・サービス・プログラムは、ブートストラップ・タスクによってメモリにロードされる最初のプログラムである。ネーム・サービスが最初にロードされるのは、本発明により、ネーム・サービスと他のサーバとの接続をするためであり、ネーム・サービスをできるだけ早く起動させることにより、他のサーバがネーム・サービスを使えるようになる。これにより、暫定的なネーム・サービスが必要なくなる。

【００６６】通常、デフォルト・ページャーはブート手順の早い段階でロードされ、メモリが十分とれるようにする。デフォルト・ページャーがあることにより、ブートストラップ・タスクによりデフォルト・ページャーに渡された最初のページング・スペースに記述されたブート・デバイス上の領域にページングが行えるようになる。その結果、比較的少ないメインメモリを持ったデータ処理システムでオペレーティング・システムが立ち上げられる。次にマスター・サーバがメモリにロードされ、これにより、ブートストラップ・タスクのローディング機能が不要になる。マスター・サーバは、それがブートストラップ・タスクによりロードされた後は、システム全体の動作を見る役をする。
【００６７】マスター・サーバが起動すると、ブートストラップ・タスクは基本的なデバイス・ドライバの役割に変わり、ブート・デバイスの入出力の役割を担って、ページングやプログラムのローディングを行う。さらに、この時点で、ブートストラップ・タスクはマスター・サーバに対して、ユーザとの連絡の手段になるコンソールの役をする。適切なシステム機能が起動されると、ブートストラップ・タスクは持っている資源を全て放して終了する。ブートストラップ・タスクは１組のメッセージを持っていて、これにより、マスター・サーバが、ブートストラップ・タスクのある操作を実行するよう要求できるようになっている。
【００６８】以上の結果、本発明により、ブート・ボリュームを使用した単純なファイル・システムが提供でき、ブートストラップ・タスクが、ブート・ボリュームに含まれているファイルにアクセスできるようになる。さらに、ブートストラップ・タスクがプログラム・ローダを提供するので、最初に必要ないくつかのプログラムがロードできる。プログラムをローディングする過程で、ブートストラップ・タスクにより、プログラムが、起動されたネーム・サービスにアクセスできるようになる。さらに、ブートストラップ・タスクが１組の簡単なデバイス・ドライバを提供するので、他のサーバが、ブート・ボリュームで記述されたブート・デバイス上にデータを読み／書きできるようになる。また、ブートストラップ・タスクは、ブート・プロセス中にサーバとオペレータないしユーザが連絡できるようにするための簡単なデバイス・ドライバを提供している。またさらに、ブートストラップ・タスクは、他のプログラムがブート・ボリュームの内容にアクセスできるように１組のファイルシステム・インターフェースを提供している。この結果、データ処理システムの実際のファイルシステムがロードされ起動されるまで、ブートストラップ・タスクはブート・ボリュームのファイルシステムの役をする。
【００６９】図１１にデータ処理システムをブートするのに使われる主要な要素をブロックダイアグラムにして示す。メモリ６００は、ＯＳカーネル６０２、ブートス

トラップ・タスク６０４、マスター・サーバ６０６、および、ハードディスク・デバイス・ドライバ６０８を持っている。ＯＳカーネル６０２とブートストラップ・タスク６０４は、ブート・プロセス中にブート・ローダによりロードされる。ブートストラップ・タスクはブート・プロセスの一環としてマスター・サーバ６０６をロードする。最初、ブート・ボリュームからの情報を使って、ファイルがハードディスク・ドライブ６１０からブートストラップ・タスクによりロードされる。

【００７０】ハードディスク・ドライブからのファイルを要求するハウスキーピング機能と他のタスクは、ＯＳカーネル６０２にメッセージを送ることにより、アクセスすることができる。この段階、即ち、十分なシステム機能がメモリ６００にロードされるまで、ＯＳカーネルはメッセージをブートストラップ・タスク６０４にまわす。主要な機能の内、特にハードディスク・デバイス・ドライバ６０８をメモリ６００にローディングすることにより、マスター・サーバ６０６は、ブートストラップ・タスク６０４とブート・ボリュームを必要とすることなく、ハードディスク・ドライブ６１０にアクセスできる。その時点で、ブートストラップ・タスク６０４は終了し、ハードディスク・ドライブ６１０のファイルをアクセスする要求は、ＯＳカーネル６０２によりハードディスク・デバイス・ドライバ６０８にまわされる。ブートストラップ・タスク６０４が扱っていた他の機能も、メモリ６００にある他のプログラムあるいはデバイス・ドライバ（図示せず）にまわされる。

【００７１】上記の具体化はハードディスクを使用したが、本発明は、データ処理システムをブート即ち初期化するのに使用できる他の媒体を使って具体化することもできる。

【００７２】上記の具体化は、１台のコンピュータ上での具体化として説明したが、本発明はネットワーク、分散処理、あるいは、マルチプロセサ・データ処理システムでも実施することができる。本発明の具体化は、ユーザ・レベルのデバイス・ドライバを持ち、Machの派生物のマイクロ・カーネルを使用するデータ処理システムで実施できるものである。Mach派生物のマイクロ・カーネルについての詳細は、「Mach: オペレーティング・システムの１つの基礎」（Richard Rashid et al.,Mach: A Foundation For Operating Systems, Proceedings of the Second Workshop on Work Station Operating Systems, pp. 109-113, IEEE Computer Society, September, 1989）に記載されている。

【００７３】

【発明の効果】上述したブート・ストラップのアーキテクチャにより、本発明は、ROMブートの終了から始まり、オペレーティング・システム環境の初期化の実行までの連続的な手段を提供することができる。

【図面の簡単な説明】

【図１】本発明が実施できるデータ処理システムを示す。

【図２】図１に示したデータ処理システムの、本発明を具体化するための主たる構成要素を示す。

【図３】本発明による、データ処理システムをブートする方法を示すフローチャート。

【図４】本発明のブート・ボリュームの構造を示すダイアグラム。

【図５】本発明のブート・ボリューム・ファイル・アンカーを示すダイアグラム。

【図６】本発明のファイルヘッダを示すダイアグラム。

【図７】本発明によるブロック・リスト・エレメントを示すダイアグラム。

【図８】本発明によるＯＳカーネルに情報を送る方法のフローチャート。

【図９】本発明による、ＯＳカーネルからブート・ストラップ・タスクに情報を渡す方法のフローチャート。

【図１０】本発明の、ブート・ストラップ・タスクによりプログラムをロードする方法のフローチャート。

【図１１】本発明により、データ処理システムをブートするのに使用する主要構成要素を示すブロック図。

【符号の説明】

５０	コンピュータ
５２	システムユニット
５４、９４	ディスプレイ装置
５６、９０	キーボード
５８、９２	マウス
６０	システムバス
６２	マイクロプロセサ
６４	数値計算コプロセサ
６６	DMAコントローラ
６８	ROM
７０	RAM
７２	CMOS　RAM
７４	メモリ・コントローラ
７６	バス・コントローラ
７８	インタラプト・コントローラ
８０	キーボード／マウス・コントローラ
８２	ビデオ・コントローラ
８４	並列コントローラ
８６	直列コントローラ
８８	ディスケット・コントローラ
９６	印刷装置
９８	モデム
１００	フロッピー・ディスクユニット
１０２	ディスク・コントローラ
１０４	ハードディスク・ユニット
２９８	ブート・ボリューム
３００	ブート・ボリューム・ファイル

17

・アンカー

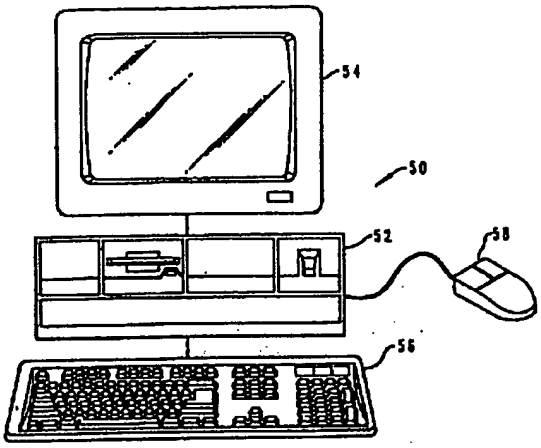
322 ファイルヘッダ

18

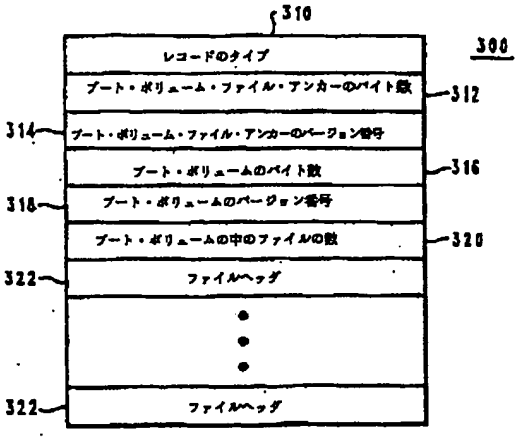
346 ブロック・リスト・エレメント

600 メモリ

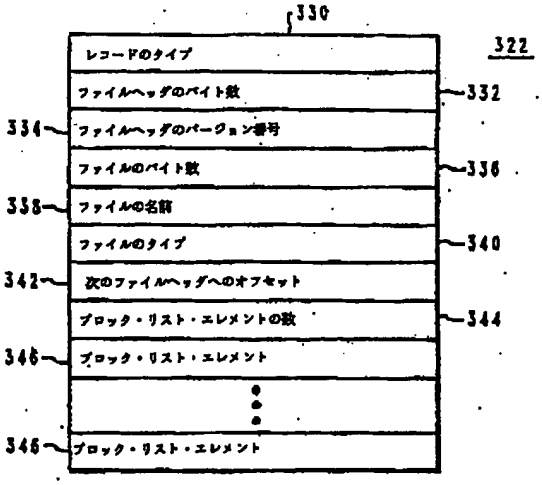
【図1】



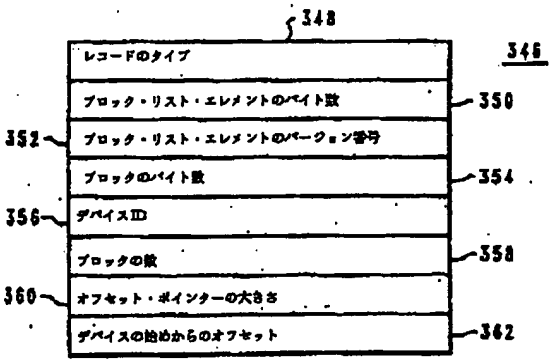
【図5】



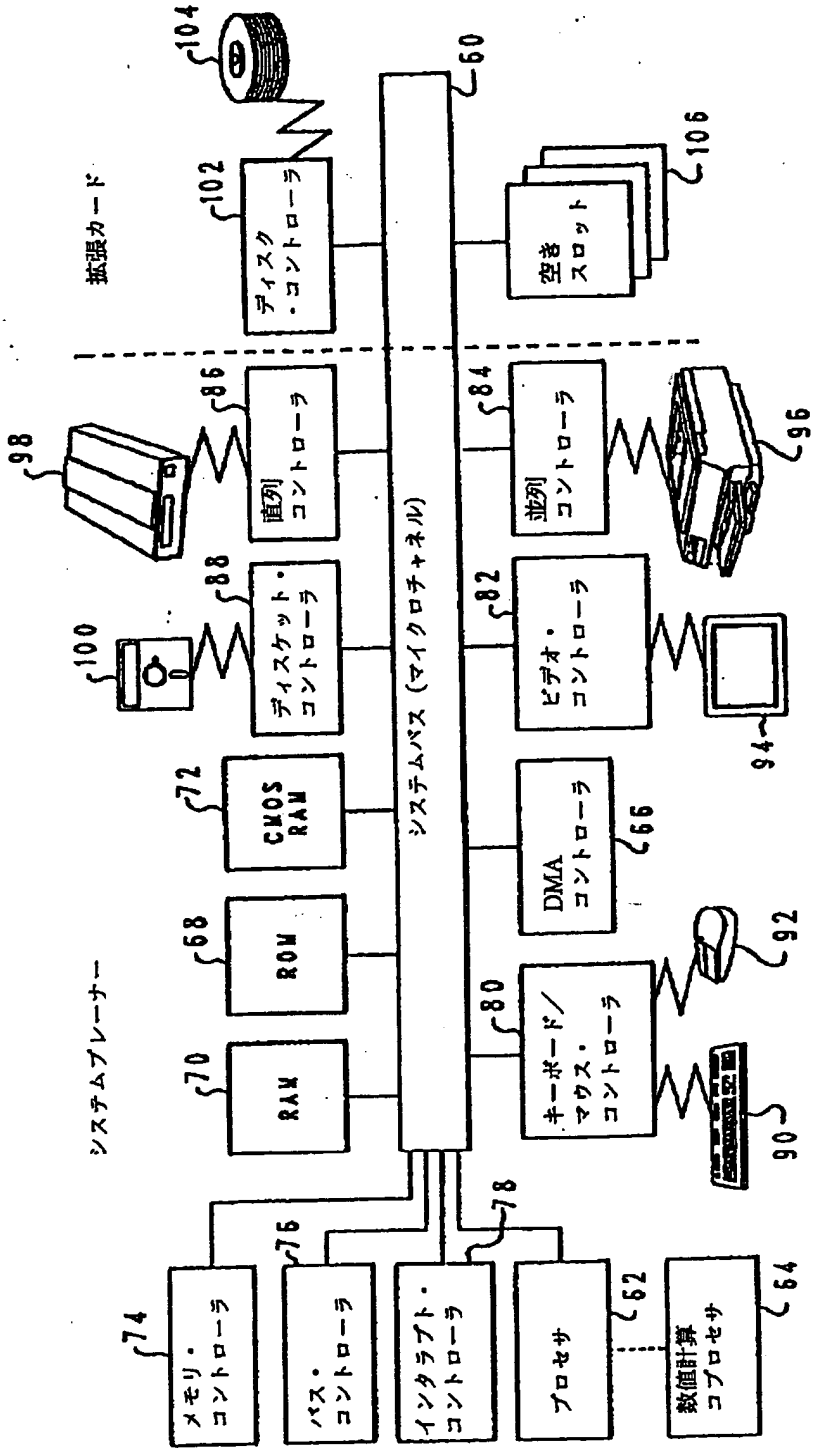
【図6】



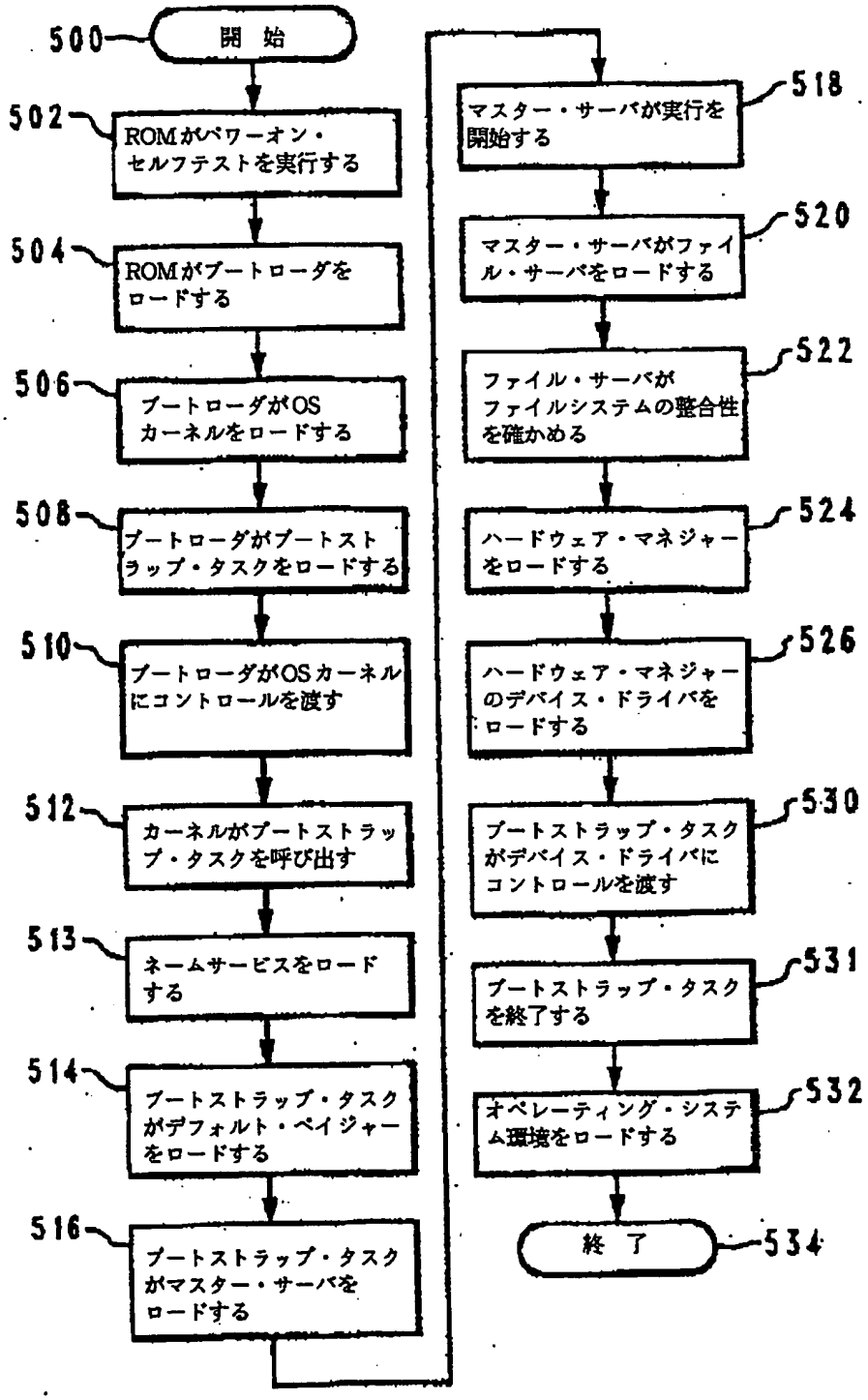
【図7】



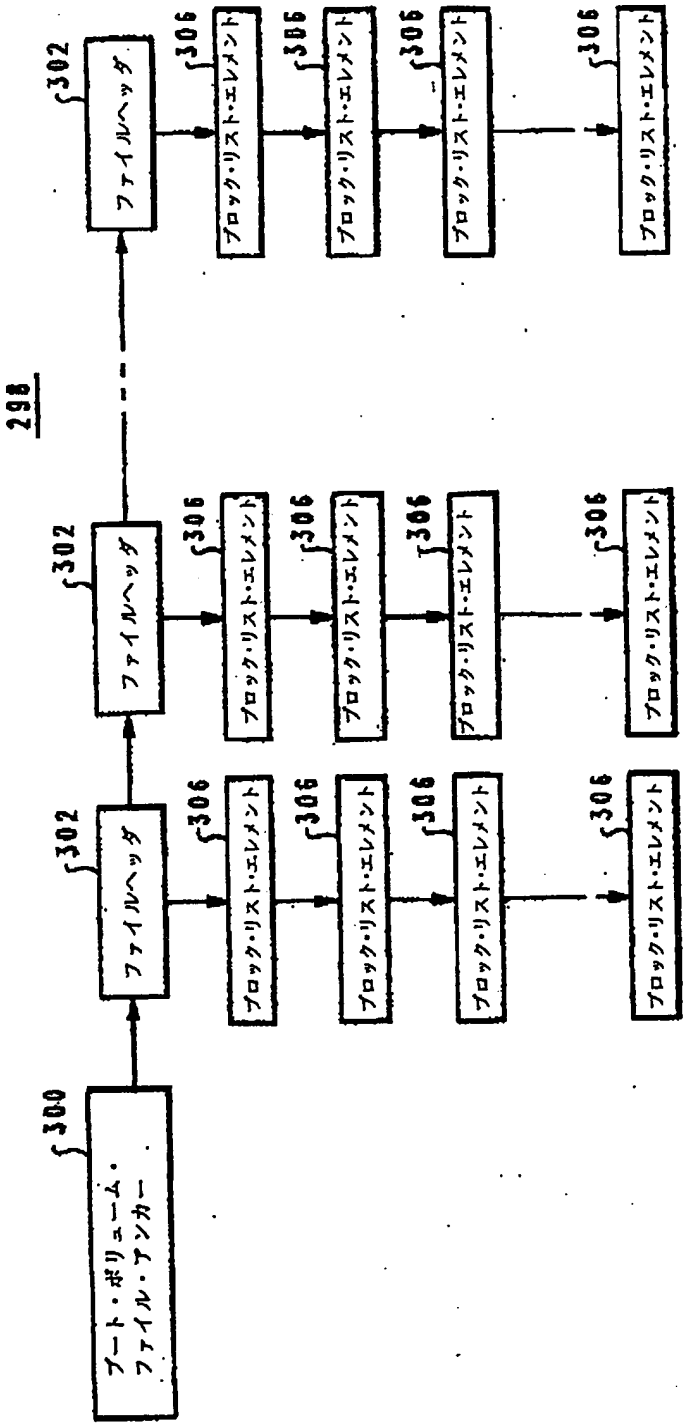
【図2】


システムプレーナー
拡張カード
システムバス（マイクロチャネル）
メモリ・コントローラ
バス・コントローラ
インタラプト・コントローラ
プロセサ
数値計算コプロセサ
RAM
ROM
CMOS RAM
ディスケット・コントローラ
直列コントローラ
ディスク・コントローラ
キーボード／マウス・コントローラ
DMAコントローラ
ビデオ・コントローラ
並列コントローラ
空きスロット
74
76
78
62
64
70
68
72
88
86
102
80
66
82
84
106
104
100
98
94
96
92
90

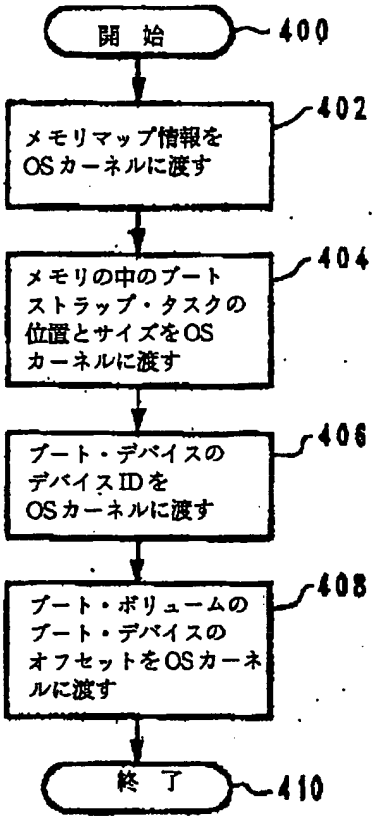
【図3】


500
開 始
502
ROMがパワーオン・セルフテストを実行する
504
ROMがブートローダをロードする
506
ブートローダがOSカーネルをロードする
508
ブートローダがブートストラップ・タスクをロードする
510
ブートローダがOSカーネルにコントロールを渡す
512
カーネルがブートストラップ・タスクを呼び出す
513
ネームサービスをロードする
514
ブートストラップ・タスクがデフォルト・ペイジャーをロードする
516
ブートストラップ・タスクがマスター・サーバをロードする
518
マスター・サーバが実行を開始する
520
マスター・サーバがファイル・サーバをロードする
522
ファイル・サーバがファイルシステムの整合性を確かめる
524
ハードウェア・マネジャーをロードする
526
ハードウェア・マネジャーのデバイス・ドライバをロードする
530
ブートストラップ・タスクがデバイス・ドライバにコントロールを渡す
531
ブートストラップ・タスクを終了する
532
オペレーティング・システム環境をロードする
終 了
534

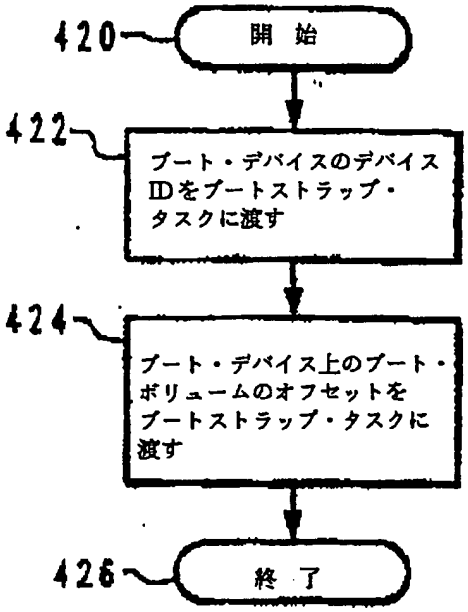
【図4】



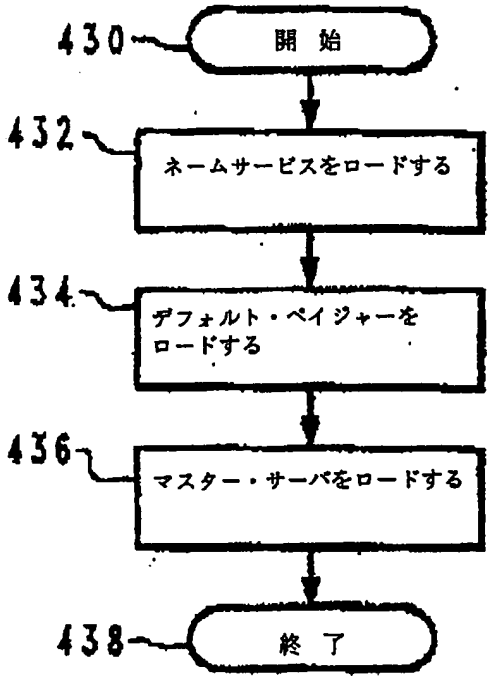
【図8】


開 始
400
メモリマップ情報をOSカーネルに渡す
402
メモリの中のブートストラップ・タスクの位置とサイズをOSカーネルに渡す
404
ブート・デバイスのデバイスIDをOSカーネルに渡す
406
ブート・ボリュームのブート・デバイスのオフセットをOSカーネルに渡す
408
終 了
410

【図9】


420
開 始
422
ブート・デバイスのデバイスIDをブートストラップ・タスクに渡す
424
ブート・デバイス上のブート・ボリュームのオフセットをブートストラップ・タスクに渡す
426
終 了

【図10】


430
開 始
432
ネームサービスをロードする
434
デフォルト・ページャーをロードする
436
マスター・サーバをロードする
438
終 了